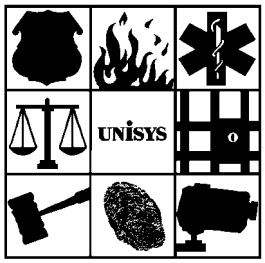
UNISYS




# Law Enforcement Message Switch (LEMS)

## *TCP/IP Interface Specification*

Version: 1.5

Date: July 30, 1999

Prepared by: Unisys Corporation
Justice & Public Safety Practice
12010 Sunrise Valley Drive
Reston, VA 20191

# Table of Contents

# Change Log

**Version 1.4**

a) Added clarification to DMPP-2020 output processing. See Section 3.3.2, item e).

**Version 1.5**

a) Updated description of client device identification to reflect implementation of capability referenced as future enhancement in previous versions of this document. See Section 3, item d).

# 1. Introduction

This Interface Specification describes the TCP/IP interface between LEMS and its data bases, workstations and external hosts which may include metros, NCIC 2000 and NLETS, assuming they adopt NCIC 2000 communications standards. This interface supports two different protocol specifications - NCIC 2000 and Datamaxx DMPP-2020[1]. The NCIC 2000 protocol is specified in the document "NCIC 2000 Interface Requirements Specification (IRS)" published by the FBI. The DMPP-2020 protocol is specified by the document "Message Header Processing in the Law Enforcement Environment using the Datamaxx Message Processing Protocol (DMPP) 2020" published by Datamaxx Applied Technologies, Inc. This document describes a specific implementation of the formats described in these documents and adapted by LEMS as its standard for messages exchanged over TCP/IP. Note that not all of the features described in the Datamaxx document are included in the LEMS implementation. The implementation of supported features is described in this document; others may be included in the future.

Where possible, agencies are strongly encouraged to use the DMPP-2020 protocol. It is much more robust than the NCIC 2000 since it supports guaranteed message delivery and provides formats for exchanging status information.

This document is written from the perspective of "clients" and "servers". Depending on the interface, LEMS may assume either role and, in most cases, LEMS will assume a combination of roles. Typically, LEMS will be a server to end-user devices such as workstations and metro hosts, and a client to data bases such as NCIC, NLETS, DMV, etc. Although message formats are the same regardless of the role, the responsibilities for establishing and maintaining the connection, as explained below, are different.

# 2. Applicable Documents

The following documents are referenced on this specification:

a) "NCIC 2000 Interface Requirements Specification (IRS)", Section 19, dated May 1997.

b) "Message Header Processing in the Law Enforcement Environment using the Datamaxx Message Processing Protocol (DMPP) 2020" (Revision 4, 9/97).

# 3. Connection Establishment and Maintenance

Regardless of the protocol, connections shall be established and maintained as follows:

a) Connectivity will be based on TCP/IP sockets created in the internet domain (AF_INET) with a communications type of TCP (SOCK_STREAM).

[1] Datamaxx Message Processing Protocol and DMPP-2020 are registered trademarks of Datamaxx Applied Technologies, Inc.

b) Servers will assume the role of connectivity server, listening for and accepting connections to a TCP/IP socket on a predefined IP address and one or more ports. The specific addresses and ports will vary from site to site.

c) Clients will be responsible for connecting to the server at a specific TCP/IP address and port. Once a connection is established, it will be kept open for the duration of the session. Both the client and server applications will send and receive data on the same connection, and should therefore implement non-blocking, asynchronous data communications.

d) There are two methods available for a client to identify itself to a server as a connection is being established. These two methods are described below. When Validation String Identification is used, it is applied after TCP/IP Address Identification occurs – it is an additional check.

   1) **TCP/IP Address Identification:** When a server receives a connection request from a client, it will attempt to match the TCP/IP address received in the request with that of a client that is allowed to connect to the requested port. If there is a match, the connection will be allowed. If there is no match, the server will close the connection immediately. If there is a match and there already exists a connection with that client at the requested port, the server will close the existing connection and allow the new one.

   2) **Validation String Identification:** Once a connection is established, the server will not allow its use by client or server applications until the client is identified. For this to happen, a message must be received from the client that contains a validation string of up to 24 alphanumeric characters (not case sensitive). When such a message is received, the server will attempt to match the string with that of a client that is allowed to connect to the requested port. If a match is made, the server will send a message to the client containing "CONNECTION ACCEPTED". Once the server sends this message, its applications can begin using the connection. The client's applications can begin using it upon receipt of the message. Both of these messages should be exchanged in the simplest manner possible and not use requests for acknowledgment. If the string received from the client doesn't match that of one of the server's expected clients, or such a message is not received within a predefined period of time (2 minutes for LEMS), or any other message is received from the client prior to sending the acceptance message, the server will immediately close the connection. If the client doesn't receive the expected response within a predefined period of time, or receives any other message from the server prior to the acceptance message, the client will immediately close the connection. LEMS will support this feature in the server role only.

e) If the client doesn't receive an expected response to a message sent to the server or it receives a negative acknowledgment, it may perform a number of retries after which it should close the connection, re-connect and try again. However, experience has shown that if a message isn't successfully delivered on the first attempt, it will probably also be unsuccessful on retries, so the most expedient measure is to close and re-open the connection immediately upon the first detection of failure.

f) If the server doesn't receive an expected response to a message sent to the client or it receives a negative acknowledgment, it may perform a number of retries after which it should close the connection and wait for the client to re-connect. Again, experience has shown that the most expedient measure is to close the connection immediately upon the first detection of failure and wait for the client to re-connect.

g) The connection may be terminated by either the client or the server at any time.

h) Using the DMPP-2020 protocol, both the client and the server can monitor the status of the connection using the status request and response functions described below. Either can terminate the connection if problems are detected. The client is responsible for re-establishing the connection if necessary. Note that in this architecture, the server is not capable of re-establishing the connection. The client, therefore, must monitor the status of the interface and ensure that connectivity is maintained when required.

## 3.1 Protocol Processing

## 3.2 NCIC 2000 Protocol

### 3.2.1 Message Structure

Messages exchanged using the NCIC 2000 protocol shall be structured as shown below. See section 19 of the NCIC 2000 IRS for further details.

[STAP] [BLK LEN] [DATA] [STOP]

| NCIC 2000 Message Structure | | |
|---|---|---|
| **Field** | **Description** | **Specification** |
| STAP | Start pattern. | 0xFF00AA55 |
| BLK LEN | Length of message from STAP to STOP, inclusive. | 2 byte unsigned integer. |
| DATA | Message text. | Format depends on message key. Maximum length 32K. |
| STOP | Stop pattern. | 0x55AA00FF |

### 3.2.2 Output Processing

Outputs sent using the NCIC 2000 protocol shall be structured as specified in the table above.

### 3.2.3 Input Processing

Inputs received using the NCIC 2000 protocol should be structured as specified in the table above. The input stream should be scanned for STAP. Once detected, BLK LEN should be determined from the next two bytes and that number of bytes read. The last four bytes should

contain STOP. If so, DATA contains a message which should be processed according to the requirements of its message key. Otherwise, the bytes should be discarded and the inputs scanned for the next occurrence of STAP.

### 3.2.4 Examples

#### 3.2.4.1 Transaction Entered by User

This example shows the message exchange resulting from a user entering a QV transaction which is sent to NCIC 2000. LEMS will respond to the entered transaction with an application level acknowledgment after the transaction is processed. Note that this example assumes a two character line terminator (e.g., CR/LF).

**The user sends a transaction to LEMS:**
[STAP] [LEN=40]
[DATA="QV.PA1234567.LIC/ABC123.LIS/KY"] [STOP]

**LEMS responds by sending an application level acknowledgment to the user:**
[STAP] [LEN=24]
[DATA="ACK 1234567890"] [STOP]

**LEMS forwards the transaction to NCIC:**
[STAP] [LEN=63]
[DATA="1L011234567890QV1 .QV.PA1234567.
LIC/ABC123.LIS/KY"] [STOP]

**NCIC responds to LEMS:**
[STAP] [LEN=69]
[DATA="1L011234567890QV1
PA1234567
NO RECORD LIC/ABC123 LIS/KY"] [STOP]

**LEMS forwards the response to the user:**
[STAP] [LEN=85]
[DATA="MSG 1234567890
1L011234567890QV1
PA1234567
NO RECORD LIC/ABC123 LIS/KY"] [STOP]

## 3.3 DMPP-2020 Protocol

### 3.3.1 Message Structure

Messages exchanges using the DMPP-2020 protocol are structured as shown below. See Datamaxx documentation for further details.

[STAP] [BLK LEN] [HDR LEN] [FC] [VAL]
[DATA LEN] [SC] [DEST] [ENC LEN] [ENC TYPE] [ENC KEY]
[DATA] [STOP]

| DMPP-2020 Message Structure | | |
|---|---|---|
| **Field** | **Description** | **Specification** |
| STAP | Start pattern. | 0xFF00AA55 |
| BLK LEN | Length of message from STAP to STOP, inclusive. | 4 byte signed integer. Maximum value 32,032. |
| HDR LEN | Length of header from HDR LEN to DEST, inclusive. | 2 byte signed integer. Always set to 16. |
| FC | Function code identifying purpose of the message. | 2 byte signed integer with 10 defined values. |
| VAL | Value used to correlate status responses with requests for status. | 4 byte unsigned integer. |
| DATA LEN | Length of message data. | 4 byte signed integer. Maximum value 32K. |
| SC | Status code. | 2 byte signed integer with 14 defined values. |
| DEST | Destination of message. Not used in current implementation. | 2 byte signed integer. Always set to 1. |
| ENC LEN | Length of encryption header. Not used in current implementation. | 2 byte signed integer. |
| ENC TYPE | Encryption request type. Not used in current implementation. | 2 byte signed integer. |
| ENC KEY | Encryption key identifier. Not used in current implementation. | 4 byte unsigned integer. |
| DATA | Message text. | Format depends on message key, maximum length 32K. |
| STOP | Stop pattern. | 0x55AA00FF |

| DMPP-2020 Function Codes | |
|---|---|
| **FC** | **Description** |
| 01 | Data message, no request for acknowledgment. |
| 02 | Data message, request acknowledgment. |
| 17 | Positive acknowledgment to data message (ACK). |
| 18 | Negative acknowledgment to data message (NAK). |
| 33 | Status request. |
| 34 | Status response. |
| 49 | Coded message 1 request. |
| 50 | Coded message 2 request. |
| 65 | Coded message 1 response. |
| 66 | Coded message 2 response. |

| DMPP-2020 Status Codes | |
|---|---|
| **SC** | **Description** |
| 01 | Successful receipt of data message (ACK). |
| 17 | Permanent error. |
| 18 | Temporary error. |
| 19 | Logical error. |
| 33 | Queried destination is available and ready. |
| 34 | Queried destination is available, but not ready. |
| 35 | Queried destination is not available and not ready. |
| 49 | Invalid function code received. |
| 50 | Invalid destination received. |
| 51 | Invalid header format or length received. |
| 52 | Function not supported. |
| 65 | Attempt to start encryption with no key definition. |
| 66 | Invalid encryption header format or length received. |
| 67 | Encryption not supported. |

### 3.3.2 Output Processing

Outputs sent using the DMPP-2020 protocol shall be structured as specified in the table above, using the following rules.

a) Outputs with FC=1 (data message, no request for ACK), 2 (data message, request ACK) or 33 (request for status) should be sent with VAL containing a unique, sequential value. The values must use the full range of the field. These outputs should also have SC=33 (available and ready).

b) Outputs with FC=17 (ACK), 18 (NAK) and 33 (status response) should be sent with VAL containing the value received in the related input and SC set to the appropriate value (see input processing).

c) Only a single output with FC=2 (data message, request ACK) may be outstanding until either a message with FC=17 (ACK) or FC=18 (NAK) is received or the application times out waiting for it. Specific implementations may impose additional rules for entering transactions such as waiting for an application level acknowledgment before sending the next message.

d) Only a single output with FC=33 (request for status) may be outstanding until either a message with FC=34 (status response) is received or the application times out waiting for it.

e) While waiting for the response to either FC=2 (data message, request ACK) or FC=33 (request for status), the following outputs should not be sent: FC=1 (data message, no request for ACK), FC=2 (data message, request ACK), or FC=33 (request for status). Note however that while waiting for the response, there is no guarantee that the next input will be the response. The next input could also be any of the following: FC=1 (data message, no request for ACK), FC=2 (data message, request ACK), or FC=33 (request for status).

### 3.3.3 Input Processing

Inputs received using the DMPP-2020 protocol should be structured as specified in the table above. The input stream should be scanned for STAP. Once detected, BLK LEN should be determined from the next four bytes and that number of bytes read. The last four bytes read should contain STOP. If so, the individual fields of the header and DATA should be identified and processed as described below. Otherwise, the bytes read should be discarded and the inputs should be scanned for the next occurrence of STAP.

If BLK LEN is valid, the following validity checks shall be made on the header fields. An invalid conditions shall result in the indicated response for the condition. A response shall be sent for only the first invalid condition detected.

| Invalid Header Conditions and Responses | |
|---|---|
| **Condition** | **Response** |
| HDR LEN=24 (encryption header present) | FC=18 (NAK)<br>SC=67 (encryption not supported)<br>VAL=received value |
| HDR LEN <> 16 or 24 | FC=18 (NAK)<br>SC=51 (invalid header)<br>VAL=received value |
| FC=49 (coded message 1 request), 50 (coded message 2 request), 65 (coded message 1 response) or 66 (coded message 2 response) | FC=18 (NAK)<br>SC=52 (FC not supported)<br>VAL=received value |
| Invalid FC | FC=18 (NAK)<br>SC=49 (invalid FC)<br>VAL=received value |
| Invalid SC | FC=18 (NAK)<br>SC=51 (invalid header)<br>VAL=received value |
| DEST <> 1 | FC=18 (NAK)<br>SC=51 (invalid header)<br>VAL=received value |
| DATA LEN <> BLK LEN - HDR LEN - 12 | FC=18 (NAK)<br>SC=51 (invalid header)<br>VAL=received value |

If the header is valid, the following responses shall be returned and actions taken.

| Responses and Actions for Valid Inputs | |
|---|---|
| **Condition** | **Response/Action** |
| FC=2 (data message, request ACK) and message received without problems | FC=17 (ACK)<br>SC=1 (ACK)<br>VAL=received value |
| FC=2 (data message, request ACK) and message received with problems | FC=18 (NAK)<br>SC=17 (permanent error), 18 (temporary error) or 19 (logical error), as appropriate<br>VAL= received value |
| FC=17 (ACK) and VAL=last value sent | Mark last message sent as delivered. |
| FC=18 (NAK) | Resend last message sent up to the maximum retry count. |
| FC=33 (status request) | FC=34 (status response)<br>SC=33 (device available and ready), 34 (available, not ready) or 35 (not available, not ready), as appropriate<br>VAL= received value |
| FC=34 (status response) and SC=33 (device available and ready) | Mark device as up |
| FC=34 (status response) and SC=34 (available, not ready) or 35 (not available, not ready) | Mark device as down |

### 3.3.4 Examples

#### 3.3.4.1 Transaction Successfully Entered by User

This example shows the message exchange resulting from a user entering a DQ transaction which is sent to DMV. LEMS will respond to the entered transaction with an application level acknowledgment after the transaction is processed. In this example, LEMS is configured such that data messages exchanged between LEMS and the user require ACKs but ACKs are not used for the exchanges between LEMS and DMV. Note that this example assumes a single character line terminator (e.g., LF).

**The user sends a transaction to LEMS:**

[STAP] [BLK LEN=51] [HDR LEN=16] [FC=2] [VAL=101]
[DATA LEN=23] [SC=33] [DEST=1]
[DATA="DQ.PA1234567.LIC/ABC123"] [STOP]

**LEMS responds by sending an ACK to the user:**

[STAP] [BLK LEN=28] [HDR LEN=16] [FC=17] [VAL=101]
[DATA LEN=0] [SC=33] [DEST=1] [STOP]

**LEMS also sends an application level acknowledgment to the user:**
[STAP] [BLK LEN=42] [HDR LEN=16] [FC=2] [VAL=201]
[DATA LEN=14] [SC=33] [DEST=1]
[DATA="ACK 1234567890"] [STOP]

**The user responds by sending an ACK to LEMS:**
[STAP] [BLK LEN=28] [HDR LEN=16] [FC=17] [VAL=201]
[DATA LEN=0] [SC=33] [DEST=1] [STOP]

**LEMS forwards the transaction to DMV:**
[STAP] [BLK LEN=51] [HDR LEN=16] [FC=1] [VAL=202]
[DATA LEN=23] [SC=33] [DEST=1]
[DATA="DQ.PA1234567.LIC/ABC123"] [STOP]

**DMV responds to LEMS:**
[STAP] [BLK LEN=66] [HDR LEN=16] [FC=1] [VAL=301]
[DATA LEN=38] [SC=33] [DEST=1]
[DATA="DQ.DMV.PA1234567.
NO RECORD LIC/ABC123"[STOP]

**LEMS forwards the response to the user:**
[STAP] [BLK LEN=82] [HDR LEN=16] [FC=2] [VAL=203]
[DATA LEN=54] [SC=33] [DEST=1]
[DATA="MSG 1234567890
DQ.DMV.PA1234567.
NO RECORD LIC/ABC123"] [STOP]

**The user responds by sending an ACK to LEMS:**
[STAP] [BLK LEN=28] [HDR LEN=16] [FC=17] [VAL=203]
[DATA LEN=0] [SC=33] [DEST=1] [STOP]

#### 3.3.4.2 Transaction Unsuccessfully Entered by User

This example shows the beginning of the same message sequence as above but LEMS responds with a NAK due to a temporary problem. The user retries sending the message and is successful.

**The user sends a transaction to LEMS:**
[STAP] [BLK LEN=51] [HDR LEN=16] [FC=2] [VAL=101]
[DATA LEN=23] [SC=33] [DEST=1]
[DATA="DQ.PA1234567.LIC/ABC123"] [STOP]

**LEMS responds by sending a NAK to the user:**
[STAP] [BLK LEN=28] [HDR LEN=16] [FC=18] [VAL=101]
[DATA LEN=0] [SC=18] [DEST=1] [STOP]

**The user resends the transaction to LEMS:**
[STAP] [BLK LEN=51] [HDR LEN=16] [FC=2] [VAL=102]

[DATA LEN=23] [SC=33] [DEST=1]
[DATA="DQ.PA1234567.LIC/ABC123"] [STOP]

**LEMS responds by sending an ACK to the user:**
[STAP] [BLK LEN=28] [HDR LEN=16] [FC=17] [VAL=102]
[DATA LEN=0] [SC=1] [DEST=33] [STOP]

### 3.3.4.3 Status Request and Response

This example shows a status request and response ("keep alive") sequence initiated by the user with both a positive response and a negative response due not being ready to process the input.

**User sends a status request:**
[STAP] [BLK LEN=28] [HDR LEN=16] [FC=33] [VAL=104]
[DATA LEN=0] [SC=33] [DEST=1] [STOP]

**LEMS sends a positive response:**
[STAP] [BLK LEN=28] [HDR LEN=16] [FC=34] [VAL=104]
[DATA LEN=0] [SC=33] [DEST=1] [STOP]

**LEMS sends a negative response**:
[STAP] [BLK LEN=28][HDR LEN=16] [FC=34] [VAL=104]
[DATA LEN=0] [SC=35] [DEST=1] [STOP]